DLCD-CCE: A Local Community Detection Algorithm for Complex IoT Networks

Xiaolong Xu, Nan Hu, Marcello Trovati, Jeffrey Ray, Francesco Palmieri, and Hari Mohan Pandey

***Abstract*—Internet of Things (IoT) refers to the complex systems generated by the interconnections among widely available objects. Such interactions generate large networks, whose complexity needs to be addressed to provide suitable computationally efficient approaches. In this article, we propose a distributed local community detection algorithm based on specific properties of community centre expansions (DLCD-CCE) for large-scale complex networks. The algorithm is evaluated via a prototype system, based on Spark, to verify its accuracy and scalability. The results demonstrate that compared to the typical local community detection algorithms, DLCD-CCE has better accuracy, stability and scalability, and effectively overcomes the problem that existing algorithms are sensitive to the location of initial seeds.**

***Index Terms*—Complex Networks, Network Dynamics, Community Detection, IoT**

I. Introduction

MANY real-world systems associated with IoT systems, can be successfully modelled as complex networks [1], where community detection can provide an insight into their topological properties. To achieve this, nodes are grouped into different communities according to the network topology, which are densely connected. On the other hand, as discussed in [2], the connections among different communities are sparse. Depending on the context, community structures are likely to have different connotations. For example, communities in social networks represent groups of people with similar characteristics [3], whereas in biological networks they reveal biological tissue with similar functions, and communities in the Web documents contain a large number of topic-related documents.

In this article, we propose DLCD-CCE, a novel distributed local community detection algorithm based on community centre expansion. Our motivation for this research is its wide applicability, as well as providing tangible benefits to a much wider scientific and analytic community. The main contributions of this work include the following:

- An efficient method for measuring node centricity, which first calculates the nodes density, and subsequently computes the weighted average of density and its neighbours as community centricity. The larger the community centrality of a node, the more important it is in the community which it belongs to.
- The design and implementation of distributed and parallelised algorithm (DLCD-CCE), based on Spark GraphX. This allows the algorithm to be easily implementable on the current main-stream big data processing platform with good scalability.

The rest of the article is organised as follows: Sections II and III discuss the relevant existing techniques and approaches. Section IV introduces the local community detection algorithm proposed in this article and its parallelisation, and Section V details the experimental results and corresponding analysis. Finally, Section VI concludes the paper by summarising the main contributions and points out the future research direction.

II. Related Work

Since the emergence of Network Theory, several community detection algorithms have been introduced. However, DLCD-CCE has specific proerties and features, which make it particularly suited for this type of tasks, as well as providing a more efficient approach. In this section, relevant existing technologies and methods are discussed.

Niu et al. [4] have proposed a new type of multi-objective approach based on label propagation algorithm (LDMGA) for community detection in dynamic networks. Based on the multi-objective genetic algorithm, the evolutionary clustering algorithm is transformed into a multi-objective optimisation problem, which not only improves the clustering quality, but also minimises the clustering drift from one time step to the successive one. LDMGA is effective in clustering, but the search speed of the genetic algorithm is slow. To obtain better clustering results, it requires multiple iterative calculations, so LDMGA is not suitable for distributed computing. In [5], the authors propose PLPIRV (Parallel Label Propagation and Incremental Related Vertices), where label propagation progress is integrated with incremental related vertices properties. Based on the communities found in the previous interval, this algorithm adjusts the communities to which the vertices belong to incrementally, and gradually analyses the changes of the network, in order to avoid the clustering of the whole network. In [6], a new game-theoretic approach towards community detection in large-scale complex networks is introduced, which is based on modified modularity. This method was developed from a modified adjacency and modified Laplacian matrices, as well as neighbourhood similarity, which can classify a given

Xiaolong Xu and Nan Hu are with Jiangsu Key Laboratory of Big Data Security Intelligent Processing, Nanjing University of Posts and Telecommunications, Nanjing, China

Marcello Trovati, Jeffrey Ray and Hari Mohan Pandey are with the Department of Computer Science, Edge Hill University, Ormskirk, UK.

Francesco Palmieri is with the Department of Computer Science, University of Salerno, Italy.

network into dense communities. In this algorithm, all the nodes in the network are participants, and their strategy is to decide which community they should belong to, according to the maximum fitness property. These methods were enhanced by utilising MapReduce. However, this approach only provides two primitives, Map and Reduce, whose functions are not as efficient as graph computing frameworks. In addition, because of frequent disk reads and writes during the calculation process on Hadoop, MapReduce is much more inefficient than Spark and other memory-based big data processing platform, as discussed in Section III.

A community detection algorithm usually requires iterative computation of global topology information of the network, and is not suitable for large-scale network processing. Moreover, the similarity between nodes is highly correlated with the distance between nodes. Therefore, local community detection which is based on local information has become an increasingly important research area in recent years. In [7], IN-LCD, a local community detection algorithm is introduced, where the local influence index for nodes is defined, and a subset of influential nodes near the source node is calculated and constructed with the index. Subsequently, the continuous expansion of the community is realised from the subset, and the whole local community is constructed through the calculation of the similarity index between nodes and community. IN-LCD overcomes the sensitive problem of local community detection to the initial node position. However, IN-LCD uses the information of nodes' one-hop and two-hop neighbours in calculating the influence index. In large networks, this way may lead to the explosion of information in the network. Based on parallel framework GraphLab, the authors in [8] have introduced an algorithm to detect the overlapping community for large-scale networks called DOCVN (Detecting the Overlapping Community algorithm based on Vital Node Expanding). In DOCVN, nodes with high PageRank value are regarded as vital nodes, and then the affiliation degree of other nodes to these vital nodes are computed. Then, kernel communities and expanding communities are identified respectively. Finally, the kernel communities and expanding communities are combined into some overlapping communities by judging where they connect tightly and realise the overlapping community identification of the large-scale network. In [9], a new multi-agent genetic algorithm (MAGA) is proposed to find local communities (LC) based on the tightest nodes rather than the given source node, which is termed as MAGA-LC. MAGA-LC first finds a set of tightest nodes and then extends the tightest nodes to get a local community. MAGA-LC performed well in finding local structure, but it is only proven to have good scalability on a single machine. In [10], a local community structure evaluation index is introduced, focusing on the selection of neighbours with specific nodes and maximising the index. However, in [11] the authors have suggested that nodes with a high degree can be regarded as the central nodes of the local network, and have a great impact on the local community, thus extending the local community from the local central nodes instead of the given nodes will result in better local community structure.

III. Experimental and Implementation of Big Data Frameworks

In order to provide the most suitable implementation for DLCD-CCE, different Big Data frameworks have been assessed and discussed. In fact, in conjunction with the theoretical and applied methods outlined in the previous sections, there has been considerable research effort into the design of frameworks, which can provide an efficient experimental approach to Big Data. This section will focus on the main frameworks widely utilised to address the computational challenges posed by Big Data.

A. Apache Hadoop

Hadoop [12] is an open-source distributed-architecture system, which applies to structured and unstructured data search, data analysis and data mining. Hadoop consists of two core parts: Hadoop Distributed File System (HDFS) and MapReduce. An HDFS-based cluster has two types of nodes operating in the master-slave pattern: a *namenode* (master) and a number of *datanodes* (slaves). MapReduce is a programming model for data processing, which is composed of two types of nodes that control the job execution process: a *jobtracker* (master) and a number of *tasktrackers* (slaves). The jobtracker coordinates all the jobs deployed on the system by scheduling tasks to run on tasktrackers. Tasktrackers execute tasks and send progress reports to the jobtracker, which keeps a record of the overall progress of each job.

B. Apache Spark

Spark [13] is a unified analytics engine for large-scale data processing. Apache Spark achieves high performance for both batch and streaming data, using a state-of-the-art DAG scheduler, a query optimiser, and a physical execution engine. Spark offers over 80 high-level operators that make it easy to build parallel apps. Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. Spark runs on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud. It can access diverse data sources. When a Spark job is submitted to the cluster, the cluster resource manager will start the driver process on a machine. The driver process is responsible for maintaining the context of the Spark job and splitting it into tasks, and then applies resources from the scheduler for these tasks.

C. Apache Flink

Apache Flink [13] is a framework and distributed processing engine for stateful computations over unbounded and bounded data streams. Flink has been designed to run in all common cluster environments, perform computations at in-memory speed and at any scale. Precise control of time and state enable Flink's runtime to run any kind of application on unbounded streams. Bounded streams are internally processed by algorithms and data structures that are specifically designed for fixed sized data sets, yielding excellent performance. Apache Flink requires compute resources in order to execute

applications. Flink integrates with all common cluster resource managers such as Hadoop YARN, Apache Mesos, and Kubernetes but can also be setup to run as a stand-alone cluster.

D. Experimental Evaluation of Hadoop, Spark and Flink

Since Hadoop will produce a lot of disk dumps in the computing process, its efficiency is far lower than Spark, Flink and other memory-based computing frameworks. HDFS and Yarn do not belong to Spark, but provide data storage and cluster resource scheduling functions for Spark respectively. Spark can automatically read the cached data in memory, avoiding the large amount of time overhead of disk I/O. This memory caching mechanism makes Spark 10-100 times faster than Hadoop in data processing speed. Graphx is an important component of distributed parallel graph computing. It provides rich programming interfaces related to graph processing, which enables users to process and analyse large-scale networks with the power of Spark cluster. Graphx provides a large number of operators for graph data processing, including attribute operators, structure operators, adjacency aggregation operators and so on. In addition, Graphx also provides the implementation of Pregel graph processing framework.

Spark and Flink both provide native connections to Hadoop and NoSQL databases, and can process HDFS data. In terms of streaming function, Flink is better than Spark and has native support for streaming because Spark processes streams in micro batch. However, Spark is the most active platform in the Apache repository, with very strong community support and a large number of contributors. In [13], the authors discuss an experimental evaluation of the above Big Data frameworks by considering various scenarios and workloads. In particular, they suggest that Spark is the most efficient framework in terms of speed, for both small and large dataset. On the other hand, Hadoop has a better performance compared to Flink except for small datasets. Furthermore, in Flink, all jobs are modelled as graphs that is are distributed on the cluster. As a consequence, Flink performance is affected by network states. For a more comprehensive experimental evaluations of the above frameworks, refer to [13].

IV. Description of the Algorithm

In this section, the main components of the proposed algorithms are introduced and discussed.

A. Main Definitions

As defined in [14], a network $G = G(V, E)$ comprises a set of nodes V and the set of edges E, which refer to their mutual relationships. In particular, local community detection refers to the detection of communities that contain all the neighbours of a node subset V_s. More specifically, let C be a set of communities in the network G, so V_s is a subset of all nodes in a network, that is $V_s \subset V$, and C_s is the smallest subset of C containing all nodes in V_s. Therefore, C_s satisfies

$$\forall v_i \in V_s, v_i \in C_s \quad \text{and} \quad \forall c \in C_s, \exists v_i \in c, v_i \in V_s.$$

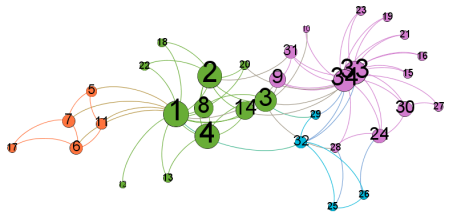

Fig. 1. The node community centricity and communities in the karate dataset

Definition 1: The relationship density of a node refers to the closeness between node itself and its neighbours.
A node with high relationship density should form more triangles with its neighbours, and links in these triangles also have a greater weight. For node v_i, v_j and v_k are the neighbours, and d_i is the degree of v_i and w_{ij} is the weight of the link (v_i, v_j), and the relationship density R_i of v_i is calculated as 1.

$$R_i = \frac{1}{d_i} \sum_{j,k} w_{jk} (w_{ij} w_{ik})^{\frac{1}{2}} \ln (d_i) \tag{1}$$

Definition 2: The community centrality of a node is a measure of its importance with respect to the community, which it belongs to.

In a community, important nodes often have close relations to other members, and the more important a node is, the higher its influence in the community. In this article, the normal distribution is used to characterise the relationship density between a node and its neighbours. The community centrality of node v_i is therefore defined as:

$$H_i = R_i \int_0^1 f(x, \sigma) dx + \frac{1}{d_i} \sum_{j=1}^{n} R_j w_{ij} \int_1^{+\infty} f(x, \sigma) dx, \tag{2}$$

where

$$f(x, \sigma) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \tag{3}$$

is the standard normal distribution, which is used to allocate weights. In the Karate dataset [15], the application of community centricity measurement in this article is shown in Figure 1, where the size of the node represents community centricity as proposed in this article, and the different colours represent different communities. It can be seen that the nodes with greater community centricity tend to be closely linked with other members in the same community.

Definition 3: The *community attractiveness* refers to the attractiveness of the community to outside nodes in the process of community centre expansion. The attractiveness of local communities to node v_i is calculated as:

$$\alpha = \frac{\sum_{j=1}^{m} \Phi(i, j)}{\sum_{k=1}^{n} \Phi(i, j)} \tag{4}$$

In (4), m is the number of nodes added to the community, n is the total number of nodes in the network, v_i is the

node in the network, v_j is the node added to the current local community, and if the link between v_i and v_j exists, $\Phi(i,j) = w_{ii}, \Phi(i,j) = 0$ otherwise.

Initially, DLCD-CCE searches nearby community centre nodes of specific seeds, eventually reaching the convergence condition, while obtaining the local community structure. The overall calculation of the algorithm is shown in Figure 2.

B. Local Community Centres Identification Via Random Walks

Traditional local community detection algorithms tend to be sensitive to seed positions. When their positions are close to community centres, high accuracy can be obtained. However, the detection accuracy is much lower when positions of seeds are on the margins of communities. Based on the random walk algorithm, we search for nearby community centres starting from the seeds within a limited number of walking steps and use the community centrality to guide the walking. Suppose the number of walking steps is limited to S, the walking process is as follows:

1) Initialise seed nodes, add a community centre tag for each node, and set a current walking step $S' = 0$;
2) If $S' \leq S$, go to step 3, otherwise go to step 5;
3) From the nodes with community centre tag, find one node from the neighbours that satisfy $H_j > H_i$ and have the max $w_{ij}H_j$. If there exists a node satisfying these conditions, go to step 4; otherwise, go to step 5;
4) Move community centre tag to the nodes found in step 3, $S' = S' + 1$, go to step 2;
5) End of walk.

At the end of the walk process, DLCD-CCE locates the nodes closer to the community centre near the initial seeds. As a result, the local community structure containing the initial seeds can be more accurately detected.

C. Community Centre Expansion

The local community detection of DLCD-CCE is based on the properties of the community centre nodes. In particular, DLCD-CCE chooses a batch of nodes in each iteration based on the community attractiveness of communities to nodes. For a particular threshold Θ, if community attractiveness α to node $v_i > \Theta$, then v_i is added to the community, otherwise, it is rejected. The steps to detect all nodes in a local community are:

1) Acquire the set community centre nodes, initial each node in this set as a single-node community respectively;
2) In the neighbours of community centres, add nodes that form triangles with community centres to the local community;
3) Aggregate all nodes in a single community into a single point, sum the weights of links from a community to the same external node as the weight of the link between the new aggregated node and the external node;
4) Add a node that is attracted to the community, and the attractiveness is greater than Θ into the community. If no neighbour node meets the conditions, go to step 6;
5) Calculate the community cohesion coefficient, if the cohesion coefficient increases, go to step 3, otherwise go to step 6;
6) End of the iteration process.

The community cohesion coefficient in step 5 is calculated as follows:

$$\omega = \frac{w_{in} + w_{out}}{w_{in}}, \tag{5}$$

where w_{in} is the sum of the weights of the inside links of the community, and w_{out} is the sum of the weights of the outside links of the community. The aggregated nodes obtained in the above process are the nodes of the local communities.

D. Distribution and Parallelisation

In order to ensure DLCD-CCE is suitable for large-scale network processing, the algorithm is parallelised based on the design idea of Bulk Synchronous Parallel (BSP) [16], so that the algorithm can run in a distributed environment.

First, the graph is decomposed into a set of triplets, such that there is a one-to-one correspondence between triplets (containing a source node, a destination node and the link between them) and links of the graph. The graph can be subsequently stored in a distributed system and the corresponding graph calculations can be decomposed for local triplets in each compute node. To implement the distributed parallelisation of the algorithm, the calculation process of the whole graph must be split into the sum of the single triplet calculation process, based on the divide-and-conquer method. In the graph decomposition method described in this article, they store only one copy for each link. However, since there are likely to be multiple links sharing the same nodes, some of them may have an original and multiple copies. After each iteration, a suitable synchronisation between the original and copies of the same node is necessary to ensure the correctness of the next iteration calculation.

1) Node Community Centrality Calculation: In a distributed computing environment, the calculation of community centrality main task includes the collection of information of links among neighbours and degrees of each node, as well as the relationship density of the neighbours of each node. In particular, the main steps are:

1) Each node is identified with its associated link weights w_{ij}, and it passes this information on to its neighbour nodes.
2) Each node collects the received information as a set and calculates the sum of the weights as the degree.
3) Each node intersects its own set with each neighbours, based on the intersection, calculate $\sum_{j,k} w_{jk}(w_{ij}w_{ik})^{\frac{1}{2}}$;
4) Calculate relationship density H^0 of each node based on (1);
5) Each node collects relationship density of neighbours and calculates the community centrality of the node based on the formula (2).

2) Identification of Local Community Centres Based on Random Walks: The process of finding local community centres includes the identification of possible community centres around given seeds. Since many networks have a small-world

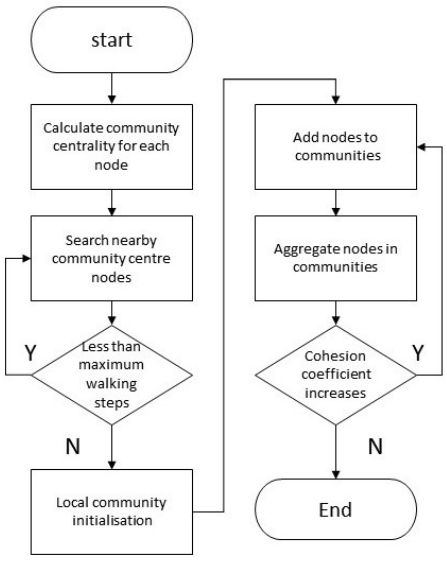

Fig. 2. Flow chart of DLCD-CCE

structure, the final community centre tag will converge to a limited number of global hub nodes in the network. As a consequence, it is necessary to define the maximum number of steps in the walk process based on the size of the corresponding networks. In a distributed computing environment, it achieves this as follows, where the maximum number of steps S:

1) Mark the flag of the initial seeds in the graph as true, set next jump direction as the node ID, and the current global walking step $S' = 0$;
2) If $S' < S$ go to step 3, otherwise go to step 6;
3) Each node receives the neighbours' ID and community centralities, discards the received message if the community flag of the current node is false;
4) On each node, if the community flag of the current node is true, select the ID with the highest $w_{ij}H_j$ in neighbours, and if the community centrality of node with the selected ID is greater than that of the current node, the next jump direction is set as the selected ID, or set as the ID of the current node;
5) Each node collects the next jump direction of its neighbours. If there is a neighbour node whose next jump direction corresponds to the ID of current node and the community flag is true, then set community flag to true, and set the community flag of the neighbour node to false. Then set $S' = S' + 1$, and go to step 2. If there is no community flag transfer in this step, go to step 6;
6) End the walk.

3) Community Centre Expansion Process: The Community centre expansion process consists of a suitable iteration of node selection and aggregation. The former is responsible for selecting eligible nodes from the neighbours of community centres and marking them. The latter will aggregate nodes that belong to the same community as new community centres.
In a distributed computing environment, the process of node selection is as follows:

1) Initialise each community centre as a single-node community and set the community ID as the ID of the current node;
2) Each node sends its ID to its neighbours, and each node collects the received ID into a set.
3) On each triplets, assume that cid is community ID of the nodes, and ccm represents community centre mark, w is the weight of the link, degree represents the degree of the nodes, and Θ is the attractiveness threshold.

Calculations performed on each triplets are shown in Algorithm 1.

Algorithm 1 Calculations performed on each triplets, as discussed in Section IV-D3.

Input: a triplet, including ID and attributes ($srcAttr$) of source node and ID and attributes ($dstAttr$) of destination node
Output: a triplet, including ID and attribute ($srcAttr$) of source node and ID and attribute ($dstAttr$) of destination node.
if $srcAttr.ccm$ and $dstAttr.ccm$ and $w/(srcAttr.degree + dstAttr.degree - w) > 0.5$ **then**
  $srcAttr.cid = \min(srcAttr.cid, dstAttr.cid)$
  $dstAttr.cid = srcAttr.cid$
else
  if $srcAttr.ccm$ and $w/dstAttr.degree > \Theta$ **then**
    $dstAttr.cid = srcAttr.cid$
    $dstAttr.ccm =$ True
  end if
else
  if $dstAttr.ccm$ and $w/srcAttr.degree > \Theta$ **then**
    $dstAttr.cid = srcAttr.cid$
    $srcAttr.ccm =$ true
  end if
end if

The process of aggregation includes the following steps:

1) Filter out triplets that community tags of two nodes are both false;
2) Filter out all triplets with only one node having true community tag, replace the node ID with the community ID for each node with true community tag, aggregate the same links, and sum their weights together;
3) Filter out triplets that community tags of two nodes are both true. Aggregate nodes with same community ID, get the number of links within each community
4) Collect result edges in the results of Step 1 and Step 2 as the edge set, and the result nodes in Step 1, Step 2 and Step 3 as the node node-set to form a new aggregated graph. After each aggregation, the cohesion coefficient of each community is calculated according to (5). If cohesion coefficient increases, then enter the next iteration, otherwise the community is no longer involved in next iterations.

V. Experimental Evaluation

Motivated by the discussion in Section III, DLCD-CCE implemention is based on Spark [17], which is an open-source general big data distributed computing platform developed by the University of California at Berkeley. Spark can cache the intermediate results of iterative calculations into memory, thus avoiding frequent disc reads and writes, and it is better suited for algorithms that require multi iterations. We deployed Spark on a cluster consisting of a master (10 MB) and three slaves (20 MB each). The network transmission rate in the cluster is between 80MB/s and 100MB/s. The version of Spark used in this experiment is 2.2.0. To verify the detection effect of the algorithm, four different scale network datasets are selected in this work as shown in Table II.

The following datasets were used: Karate [18]; Jazz [19]; Jazz musicians' cooperation network; and Polblogs [20] blog network in 2004 US presidential election. The sizes of the datasets is shown in Table I. In addition, in this article seven different data sets are used to verify the scalability of DLCD-CCE, as shown in Table II.

Data Set	Nodes	Edges
Karate	34	78
Jazz	198	2742
Polbogs	1222	16714

TABLE I
DATA SETS TO VERIFY SCALABILITY OF DLCD-CCE.

Data Set	Nodes	Edges
Karate	34	78
Jazz	198	2742
Polbogs	1222	16714
Email-Eu [21]	1005	25571
CA-HepPh [22]	12008	118521
dblp [23]	317080	1049866
youtube	1134890	2987624

TABLE II
FURTHER DATA SETS IN ADDITION TO TABLE I.

A. Evaluation and Experimental Results

Precision, recall and comprehensive index F are used to evaluate the effectiveness of DLCD-CCE. In order to compare accurately the detection results of various algorithms, community detection results of Louvain [24] is used as a test set, as it is a globally recognised community detection algorithm with high accuracy.

The definition of the various evaluation indexes are:

$$P_{re} = \frac{|C_D \cap C_R|}{|C_D|}, \tag{6}$$

$$R_{ec} = \frac{|C_D \cap C_R|}{|C_R|}, \tag{7}$$

$$F = \frac{2P_{re}R_{ec}}{P_{re} + R_{ec}}, \tag{8}$$

where C_D is the set of nodes in the local communities detected by the algorithm, and C_R is the complete set of nodes in the local communities containing the seeds. In particular, precision and recall quantify the proportion of correctly detected nodes in all detected nodes, and in the complete set of local community nodes, respectively. Finally, F is the comprehensive index, and higher values are associated with better performance.

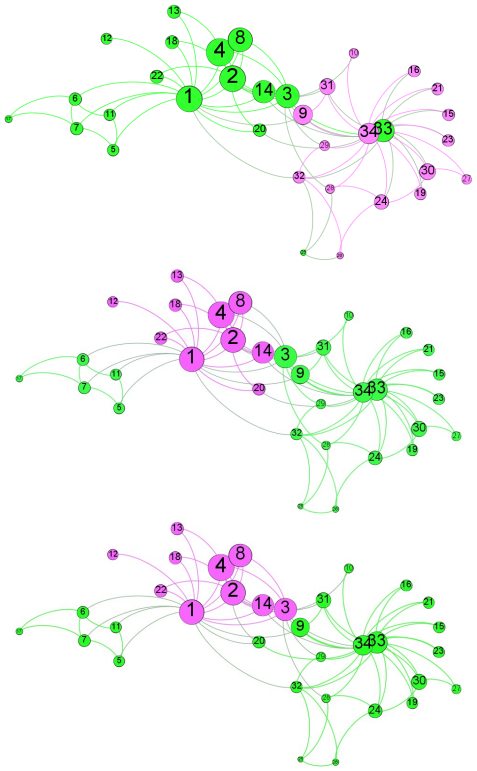

Fig. 3. The Leaf node (id 28), the Inter node (id 22) and Hub node (id2), respectively

B. DLCD-CCE Accuracy

In order to initiate the accuracy evaluation of DLCD-CCE, the different nodes are divided into three categories: Leaf, Inter and Hub, based on the last 10%, $10\% - 80\%$ and the top 10% ranking. Subsequently, randomly seeds in each ranking interval are selected, and the algorithm is evaluated according to different selection strategies. In this experiment, the attractiveness threshold, Θ, is set to 0.8. In the Karate dataset, the Leaf node (node ID 28), the Inter node (ID 22), and the Hub node (ID 2) are selected as seeds for evaluation. The detection results are shown in Figure 3, which compare the results with the detection results of Clauset [10], LWP [25], LMD [11] and LCD-NJ [26]. Due to the randomness of the position of the selected seeds, for each data set, DLCD-CCE is executed three times, and the average value is considered as the final results. Figures 4 and 5 depict the Leaf, Inter and Hub nodes in the Karate dataset, respectively.

The comparison of the experimental results for the he Jazz dataset is shown in Figure 6.

The comparison of the experimental results related to the Polblogs dataset is shown in Figure 7. It can be seen that DLCD-CCE has a good detection effect for different node

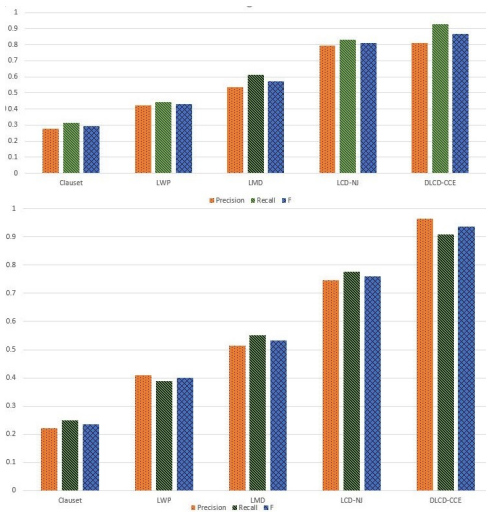

Fig. 4. (b) Karate-Inter

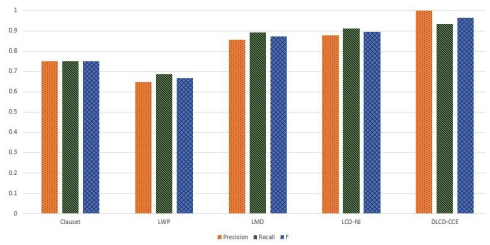

Fig. 5. (c) Karate-Hub

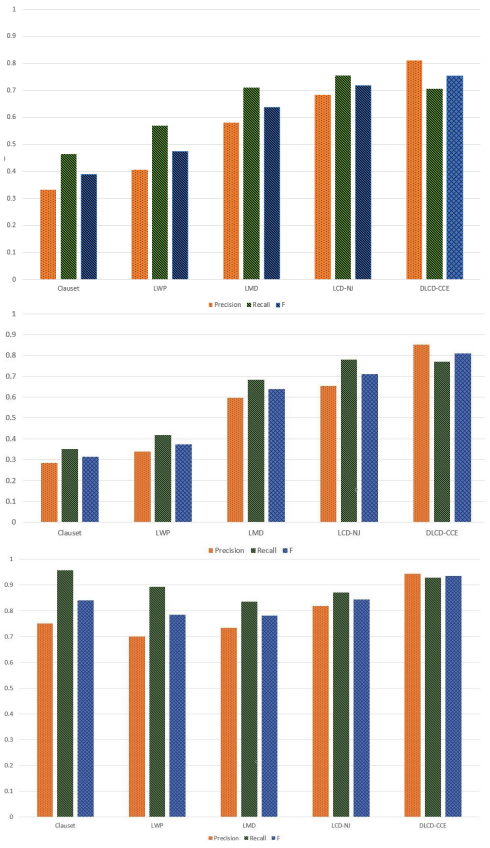

Fig. 6. The comparison for Jazz-Leaf, Jazz-Inter and Jazz-Hub, respectively

selection strategies and the results are less fluctuating. DLCD-CCE also effectively overcomes the problem that most existing local community detection algorithm is sensitive to seed positions. Furthermore, for different scales of networks the general accuracy can obtain a high value, which shows that DLCD-CCE can adapt to different scales of networks.

Finally, in order to test the expansibility of the algorithm, more experimental evaluation was carried out on 7 different-scale datasets, including Karate, Jazz, Polblogs, Email-Eu, CA-HepPh, dblp and youtube.

Six executors were set, and each of them was allocated 5GB of memory with 6 CPU cores. The execution time of the algorithm is shown in Figure 8. It can be seen that in small networks, the execution time of DLCD-CCE changes slowly, for the start-up time is much longer than the data processing time. When the number of network nodes reaches one million, the execution time of the algorithm increases linearly, and the distributed computing shows its effect of acceleration. It shows that DLCD-CCE has good scalability and can be used to deal with large-scale networks.

1) DLCD-CCE Scalability: The Spark-related parameters of the datasets described in Table II, include 6 executors, 5GB memory for each executor, and 6 total CPU cores allocated. Figure 9 depicts the execution time of DLCD-CCE

It can be seen that DLCD-CCE is significantly accelerated in distributed clusters, but with the increase of CPU cores, the acceleration is gradually slowing down. This is mainly

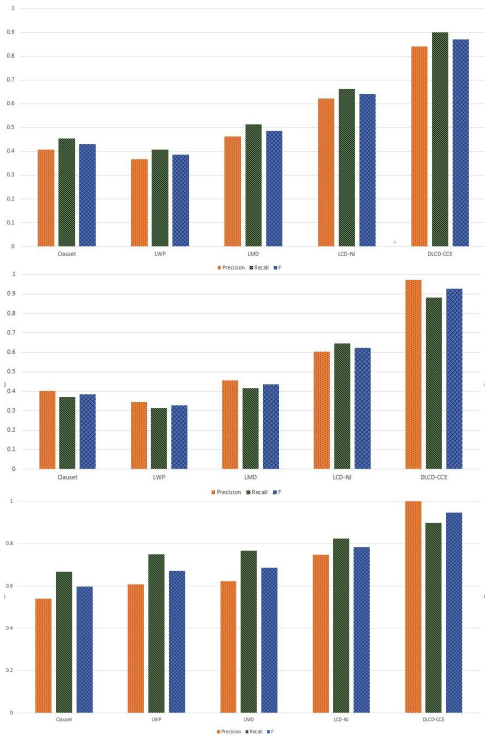

Fig. 7. The Polblogs-Leaf, Inter and Hub nodes, respectively.

due to the vertex-cut strategy of storing graphs in spark. If

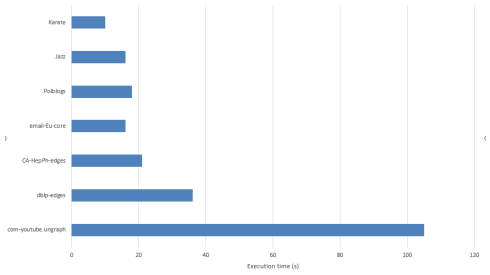

Fig. 8. Execution time of experiments on 7 different-scale datasets.

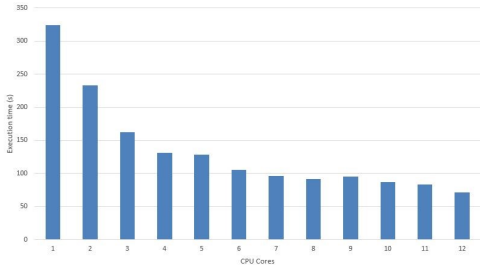

Fig. 9. The execution time of DLCD-CCE for scalability purposes.

the number of partitions is too large, network connectivity between partitions also increases.

VI. Conclusions

Community detection is widely investigated and applied area within complex network analysis. However, existing algorithms are not suitable for a distributed environment, which leads to various computational issues when analysing large-scale networks. In this article, we propose a new method to identify, assess and rank the importance of nodes in a community. Based on this method, we have designed DLCD-CCE, a local community detection algorithm. In the identification of any suitable community, DLCD-CCE only utilises the local network topology information, which makes the algorithm suitable for a distributed computing environment. Experimental results clearly demonstrate that for different scales of network, compared with the existing local community detection algorithm, DLCD-CCE has a higher accuracy and good scalability, as well as providing a more efficient method for measuring node centricity. Furthermore, its design and implementation is based on Spark GraphX, which allows a seamless and scalable implementation on the current mainstream big data processing platforms.

Future research will specifically focus on the dynamical properties related to large-scale networks, with the main objective of detecting communities in near real-time. In particular, the investigation of the topological properties of such networks would enable a better assessment of their overall behaviour, which will be integrated with DLCD-CCE to provide enhanced results.

References

[1] S. Misra, R. Barthwal, and M. Obaidat, "Community detection in an integrated internet of things and social network architecture," 12 2012, pp. 1647–1652.

[2] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *PNAS; Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[3] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3, pp. 75–174, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0370157309002841

[4] X. Z. Niu, W. Y. Si, and F. She, "Evolutionary community detection in dynamic networks," in *Journal of Software*, vol. 28, no. 7, 2017, pp. 1773–1789.

[5] G. Li, K. Guo, Y. Chen, L. Wu, and D. Zhu, "A dynamic community detection algorithm based on parallel incremental related vertices," in *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)(*, Mar. 2017, pp. 779–783.

[6] P. Chopade and J. Zhan, "A framework for community detection in large networks using game-theoretic modeling," *IEEE Transactions on Big Data*, vol. 3, no. 3, pp. 276–288, Sep. 2017.

[7] Z. C. Chen, C. Chang H., R. Y. Hunag, H. T. Yu, and Y. Liu, "A local community detection method using expansion of influential nodes set," *J. Journal of Xi'an Jiaotong University*, vol. 50, no. 4, pp. 41–47, 2016.

[8] S. Y. Wang, Y. H. Dong, Z. C. Li, H. H. Chen, and J. B. Qian, "The identification of overlapping communities in large-scale complex networks," *J.Acta Electronica Sinica*, vol. 43, no. 8, pp. 1575–1582, 2015.

[9] P. Wang and J. Liu, "A multi-agent genetic algorithm for local community detection by extending the tightest nodes," *proc.IEEE Evolutionary Computation*, pp. 3215–3221, 2016.

[10] A. Clauset, "Finding local community structure in networks," *J.Physical Review E*, vol. 72, no. 2, 2005.

[11] Q. Chen, T. Wu, and M. Fang, "Detecting local community structures in complex networks based on local degree central nodes," j.physica a," *Statistical Mechanics & Its Applications*, vol. 392, no. 3, pp. 529–537, 2013.

[12] T. White, *Hadoop: The Definitive Guide*, 1st ed. O'Reilly Media, Inc., 2009.

[13] W. Inoubli, S. Aridhi, H. Mezni, M. Maddouri, and E. M. Nguifo, "An experimental survey on big data frameworks," *Future Generation Computer Systems*, vol. 86, pp. 546–564, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17327450

[14] M. E. J. Newman, "The structure and function of complex networks," *J.SIAM Review*, vol. 45, no. 2, pp. 167–256, 2003.

[15] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015. [Online]. Available: http://networkrepository.com

[16] L. G. Valiant, "A bridging model for parallel computation," *J. Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[17] M. Zaharia, "An architecture for fast and general data processing on," *Large Clusters" M. Morgan & Claypool*, 2016.

[18] "Zachary karate network dataset-konect" nov. 2013," Nov. 2013. [Online]. Available: http://konect.unikoblenz.de/networks/ucidata-zachary

[19] "Jazz musicians network dataset – KONECT," Apr. 2017. [Online]. Available: http://konect.uni-koblenz.de/networks/arenas-jazz

[20] "Blogs network dataset – KONECT," Apr. 2017. [Online]. Available: http://konect.uni-koblenz.de/networks/moreno_blogs

[21] H. Yin, A. R. Benson, J. Leskovec, and F. G. David, "Local higher-order graph clustering,"," *proc. 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 555–564, 2017.

[22] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *J.ACM Transactions on Knowledge Discovery from Data (ACM TKDD*, vol. 1, no. 1, p. 2007, 2007.

[23] L. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *J. Knowledge and Information Systems*, vol. 42, no. 1, pp. 181–213, 2015.

[24] V. D. Blondel, J. L. Guilaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," j.journal of statistical mechanics," *Theory and Experiment*, vol. 2008, no. 10, pp. 155–168, 2008.

[25] F. Luo, J. Z. Wang, and E. Promislow, "Exploring local community structures in large networks" proc.acm," *International Conference on Web Intelligence*, pp. 233–239, 2006.

[26] T. Wang, Y. Liu, and Y. Xi, "Detection of local community structure of complex networks based on core nodes jumping," *J. Journal of Shanghai Jiao Tong University*, vol. 49, no. 12, pp. 1809–1816, 2015.